1. INTRODUCTION

Pour ce projet, vous devrez implémenter un simple jeu vidéo de tactique tour par tour en 2D, en suivant les instructions données dans ce document.

Les objectifs du projet sont:

- Mettre en pratique les concepts de POO vus en cours
- Gagner de l'expérience en programmation en Python
- Gagner de l'expérience en conception d'algorithmes
- Apprendre à collaborer sur des projets de programmation

L'évaluation du projet se fera à partir de trois éléments:

- Les livrables:
  - Un rapport de 2 pages présentant les fonctionnalités implémentées dans votre jeu, et la justifications des choix de conception du diagramme de classes UML.
  - Un diagramme de classes UML modélisant le projet
- Une présentation orale suivie de questions
- Votre dépôt Github

Planning:

- 01/11: Annonce des groupes
- 13/12: Rendu du projet (livrables et dépôt Github)
- Semaine du 16/12: Examen oral

Le programme que vous devez réaliser est un jeu en 2D se basant sur la bibliothèque pygame. Cette bibliothèque permet de créer facilement une interface graphique dans laquelle dessiner des objets en 2D. En plus de ce document présentant le projet, vous aller avoir accès à un code de départ. Ce code peut vous servir de point de départ sur l'utilisation de la bibliothèque pygame, et notamment sur la gestion des événements tels que les appuis sur les touches du clavier.

Attention, le code de départ est très loin de fournir une implémentation satisfaisante du jeu. Son rôle est vraiment juste d'illustrer ce que l'on peut faire grâce au module pygame. Le code de départ ne comporte que deux classes. Vous êtes évidemment encouragés à ajouter d'autres classes, attributs et méthodes, de manière à correspondre au diagramme de classes UML que vous allez concevoir. Si à la fin du projet vous n'avez que trois ou quatre classes dans votre programme, c'est très mauvais signe.

2. Fonctionnement du jeu

Le jeu se joue à au moins deux joueurs. Chaque joueur dispose de plusieurs unités qu'il peut contrôler. Pendant son tour, le joueur contrôle une à une chacune de ses unités, et effectue deux actions:

- Déplacer l'unité: le joueur déplace l'unité sur une nouvelle case (ou potentiellement reste sur la même case)
- Utiliser une compétence: le joueur choisit la compétence de l'unité à utiliser pour ce tour (ou potentiellement n'utilise aucune compétence)

Chaque unité dispose d'un certain nombre de points de vie. Lorsque les points de vie d'une unité est à 0, l'unité disparaît du jeu. Lorsqu'un joueur n'a plus d'unité, il a perdu et la partie s'arrête.

La vidéo donnée en exemple montre ce à quoi le jeu pourrait ressembler à la fin de votre projet. Le jeu que j'ai conçu pour cet exemple utilise des unités et des compétences inspirées du jeu vidéo Valorant, les unités ont une vision réduites et ne peuvent voir les unités ennemies que lorsqu'elles sont dans leur champs de vision. Vous êtes bien entendu invités à vous inspirer de ce qui vous plaît, ne vous contentez pas de recopier cet exemple. Vous pouvez imaginer des mécaniques complètement différentes, tant que le jeu que vous réalisez implémente un nombre minimum de fonctionnalités. Nous reviendrons plus tard sur les critères d'évaluation.

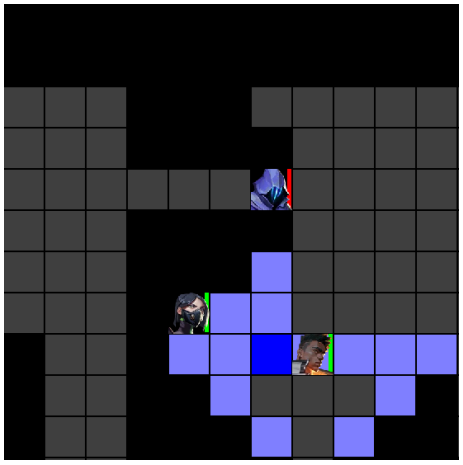
3. Interface graphique

L'implémentation de l'interface graphique utilise pygame. Vous pouvez importer cette bibliothèque au début de vos fichiers, et l'utiliser pour dessiner des objets. Le code de départ vous montre comment représenter une grille et dessiner des unités avec des cercles. Vous pouvez parcourir la documentation de pygame pour trouver des méthodes qui répondent mieux à vos besoins.

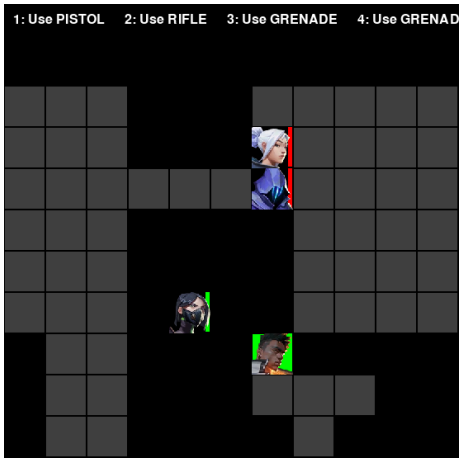
Le jeu vidéo de tactique tour par tour devra se dérouler sur un terrain correspondant à une grille en deux dimensions, comme représenté sur les captures d'écran données en figure 1. Les cases du terrain peuvent avoir différentes propriétés. Par exemples, certaines peuvent être traversables, et d'autres non, comme dans l'exemple présenté. On peut aussi imaginer des cases avec d'autres propriétés, par exemple jouant sur la visibilité des unités sur les cases, sur leur vitesse de déplacement, ou sur l'accessibilité en fonction du type d'unité (par exemple unités qui peuvent aller sur l'eau, ou dans les airs).

Chaque type de case devra être représenté d'une manière différente dans l'interface graphique, soit par une couleur associée, soit par une image. Dans les captures d'écran données en exemple, les murs sont représentés par des cases en gris foncé, que les unités ne peuvent pas traverser. Votre interface graphique devra également faire apparaître les unités présentes sur la grille de jeu, par exemple avec une image associée à chaque unité, comme dans les captures d'écran données en exemple. Il peut également être judicieux de faire figurer la barre de vie ou d'autres informations sur la miniatures. Enfin, l'interface graphique devra pouvoir permettre au joueur de choisir le déplacement d'une unité (figure 1a), choisir la compétence à utiliser (figure 1b), et la cible de la compétence (figures 1c à 1e).

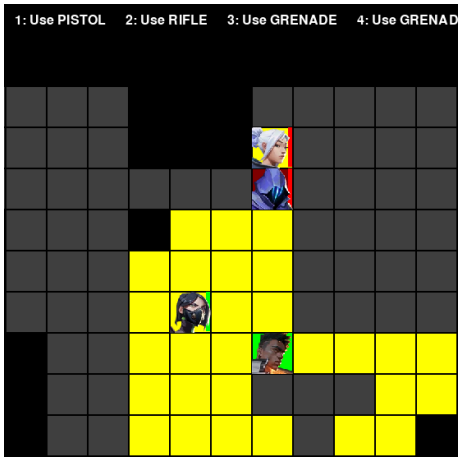
Le code de départ vous montre notamment comment gérer les événements d'appui sur les touches du clavier. Votre jeu devra être jouable au clavier, en utilisant les touches directionnelles, une touche pour valider (par exemple la barre espace), et potentiellement les touches numériques pour sélectionner des capacités, et une touche pour revenir en arrière.



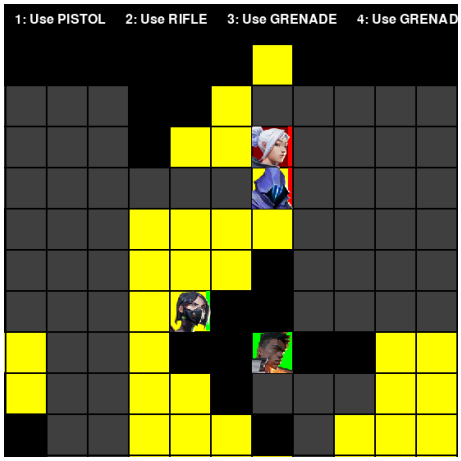
(A) Affichage des mouvements possibles (bleu pâle) et du curseur de sélection du mouvement (bleu vif).



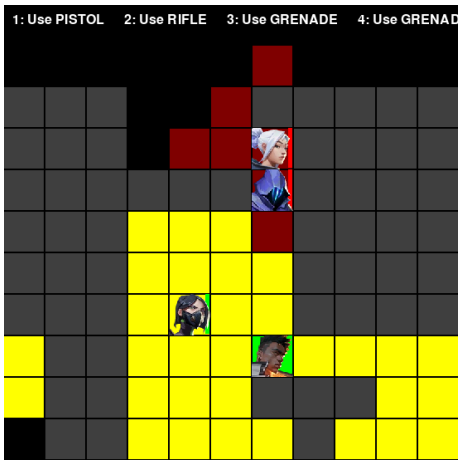
(B) Après le mouvement, un nouvel ennemi est visible. Le jeu affiche les compétences de l'unité disponibles.



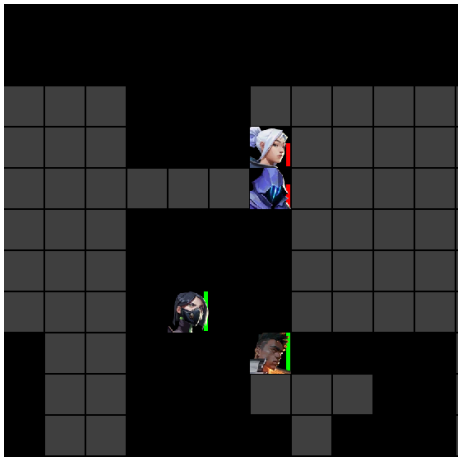
(C) Affichage des cibles possibles avec la compétence 1. Le joueur peut déplacer le curseur en rouge.



(D) Affichage des cibles possibles avec la compétence 2, qui a une portée comprise entre 3 et 7.



(E) Affichage des cibles possibles avec la compétence 3, qui fait des dégâts sur toute la zone en rouge.



(F) Des dégâts sont infligés aux deux unités ennemies (leur barre de vie diminue).

FIGURE 1. Déroulement d'un tour complet sur une unité.

4. Fonctionnalités de bases

Pour ce projet, vous devrez implémenter les fonctionnalités suivantes dans votre jeu:

- Implémentation de différentes unités: votre jeu doit comporter des unités qui ne sont pas identiques (au moins trois types d'unités). Chaque type d'unité est caractérisé, entre autre, par:
  - Un nombre de points de vie
  - Une statistique d'attaque
  - Une statistique de défense
  - Une statistique de vitesse
- Implémentation de différentes compétences: les unités peuvent posséder des compétences différentes (au moins trois types de compétences, à vous de les définir). Une compétence est par exemple une attaque sur une cible. Les compétences peuvent être caractérisées par leur puissance, leur portée, leur zone d'effet, leur précision, etc. Vous pouvez aussi imaginer des compétences ayant d'autres rôle que d'attaquer une unité ennemie (exemples: soigner une unité, faire un écran de fumée pour cacher la visibilité comme dans la vidéo en exemple).
- Implémentation de différentes cases: le terrain est composé de cases pouvant avoir des propriétés différentes (au moins trois types de cases, à vous de les définir).
- Implémentation du déplacement des unités: le joueur doit pouvoir déplacer les unités. Différentes unités peuvent avoir différentes vitesses de déplacement, et donc parcourir plus ou moins de cases pendant leur tour.
- Implémentation de la portée des compétences: différentes compétences peuvent avoir des portées différentes. La cible de la compétence doit se trouver à portée de l'unité qui utilise la compétence.
- Implémentation de la zone d'effet des compétences: différentes compétences peuvent avoir des zones d'effet plus ou moins étendues. Dans l'exemple donné, certaines compétences visent une seule case (figure 1c), alors que d'autres visent une zone étendue (figure 1e).
- Implémentation du calcul des dégâts: certaines compétences peuvent retirer des points de vie si elles ciblent une unité ennemie. Le calcul des points de vie perdus dépend de:
  - La puissance de la compétence
  - La statistique d'attaque de l'unité qui lance la compétence
  - La statistique de défense de l'unité qui est ciblée par la compétence

5. Fonctionnalités supplémentaires

En plus des fonctionnalités de bases, vous devrez choisir quelques fonctionnalités supplémentaires à implémenter dans votre jeu. Voici quelques exemples de fonctionnalités supplémentaires que vous pouvez implémenter. Vous pouvez bien évidemment imaginer d'autres fonctionnalités qui ne sont pas dans cette liste.

- Précision, esquive, coups critiques: les unités pourraient disposer de statistique d'esquive, permettant d'éviter certaines compétences dont ils sont la cible. Les compétences ou les unités pourraient avoir une statistique de précision (probabilité de réussir le coup) et de coup critique (probabilité de faire plus de dégâts).
- Systèmes de faiblesses: une mécanique populaire dans les jeux de stratégie est de rendre les compétences plus ou moins efficaces sur certains types d'unité (comme par exemple dans les jeux Pokémon, les attaques de type eau sont plus efficaces sur les Pokémons de type feu, etc.).
- Différents types de mobilité: certaines cases peuvent être accessibles uniquement par des unités spécifiques. On peut penser par exemple à des unités volantes qui peuvent passer au dessus des murs, ou à des unités pouvant nager et donc passer à travers des cases d'eau.
- Objets que l'on peut ramasser sur le terrain, qui peuvent être utilisés par l'unité les ayant ramassés. L'unité possède alors une nouvelle compétence "utiliser l'objet".
- Visibilité des unités ennemies: les unités ennemies ne sont pas visibles par défaut sur la grille de jeu. Elles deviennent visibles uniquement lorsqu'une unité du joueur entre en contact visuel direct avec l'unité ennemie. C'est par exemple le cas dans les captures d'écran données, on voit dans la figure 1b qu'un nouvel ennemi est visible une fois que l'unité du joueur a bougé.
- Compétences pouvant modeler le terrain: on peut imaginer des compétences permettant de modifier les cases du jeu, par exemple de créer des murs, de créer des zones permanentes de dégâts, etc.
- Schémas de zone d'effet: certaines compétences pourraient avoir une zone d'effet en ligne verticale ou horizontale, ou encore avec d'autres formes de votre choix.
- Implémentation d'une IA pour l'adversaire: selon l'intelligence de l'IA, cette fonctionnalité peut compter pour deux fonctionnalités supplémentaires.

Vous pouvez imaginer d'autres fonctionnalités pertinentes pour votre jeu. Dans mon cas, j'ai par exemple ajouté une mécanique qui rend les compétences plus imprécises si l'unité s'est déplacée au cours du tour (c'est un comportement classique dans les jeux de shoot comme Valorant). Si vous avez des idées qui ne sont pas dans la liste, n'hésitez pas à les implémenter et à l'indiquer dans votre rapport, ça comptera dans la note finale.

6. Evaluation

Vous serez évalués sur plusieurs aspects:

6.1. **Programme de jeu.**

- **4 points**: Fonctionnalités du jeu:
  - Est-ce que les fonctionnalités de bases sont bien implémentées? Vous devez implémenter toutes les fonctionnalités de bases listées au dessus.
  - Est-ce que vous avez implémenté des fonctionnalités supplémentaires? Vous devez implémenter **au moins deux fonctionnalités supplémentaires** de votre choix. Si vous en implémentez plus, vous pourrez avoir jusqu'à un point bonus.
- **2 points**: Interface graphique: est-ce que l'interface graphique est bien soignée? Est-ce que le jeu est beau et agréable à jouer?

6.2. **Rapport et diagramme de classes.**

- **2 points**:
  - Le diagramme de classes doit comporter **au moins deux relations d'héritage**. Le rapport doit justifier pourquoi ces relations sont pertinentes.
  - Le diagramme de classes doit comporter **au moins deux autres relations** (association simple, composition ou agrégation). Le rapport doit justifier pourquoi ces relations sont pertinentes.
  - Le diagramme de classes doit comporter **au moins une classe abstraite**. Le rapport doit justifier pourquoi cette classe est abstraite.
- **2 points**: Justesse du diagramme de classes. Est-ce que les classes, attributs, méthodes et relations sont bien représentés?
- **1 point**: Qualité et clarté du rapport. En plus de justifier les choix de conception dans votre diagramme de classes, le rapport doit aussi indiquer comment les membres du groupes se sont divisé le travail, et présenter les fonctionnalités supplémentaires qui ont été implémentées dans le jeu.

6.3. **Dépôt Github et qualité du code.**

Chaque groupe devra créer un dépôt Github public **dès le début** du projet, et y contribuer tout au long du projet. Pour l'évaluation, on regardera si votre dépôt démontre bien votre utilisation régulière de l'outil Git. Vous devez régulièrement déposer vos avancées sur votre dépôt: une bonne habitude est de faire un commit pour chaque fonctionnalité que vous implémentez ou chaque bug que vous corrigez. Tous les membres du groupe doivent participer au dépôt Github. Attention, tout le groupe est pénalisé si la participation au dépôt Git est très hétérogène. Ce n'est pas un concours pour savoir celui ou celle qui en fait le plus, le but est de vous encourager à vous répartir les tâches de manières équitable et à vous entraider.

- **2 points**: Partage équitable des tâches et participation homogène des différents membres du groupes, visible sur le dépôt Github (note commune à tout le groupe).
- **2 points**: Participation individuelle au projet (note individuelle).
- **1 point**: Organisation, lisibilité, commentaires, documentation du code.
- **1 point**: Adéquation entre le diagramme de classes UML et l'implémentation en Python.

6.4. **Oral et réponses aux questions.**

- **1 point**: Qualité de la présentation de votre projet.
- **2 points**: Qualité des réponses aux questions.